ToDoList with useState Hook

We will be creating simple ToDoList that allows user to add list, mark the task complete and remove the task from task list.

Step 1

We will create a functional component, Todolist.js
This component will be responsible for rendering the list of todolist.

So we will import useState hook from react.

```
import React, { useState } from "react";
```

We will have useState hook to allow us to store our input in a state.

```
const [todos, setTodos] = useState([]);
```

So here, todos is the name of the state and setTodos will be function to be done on our state. Our data will contain many inputs entered by user so we will use an array to store all of those inputs. So we will use map() method to set all items in our array to become list items.

```
<div className="container">
<h1>Todo List</h1>
<ul>
{todos.map((todo) => (
<li>
{todo}
</li>
)
)}
</ul>
</div>
```

We are actually running a loop for each array items such a way everytime component renders, it runs all items inside on array to display as unordered list items.

Now if we add anything to our empty array of todos, we can see it renders to get display as unordered list items.

We can style our UI to show list items to look better.

Step 2

Now we will create another component, call it ‘TodoForm’.
This component will allow user to add new todos to the list.

So first we will simply create a input form .

```
Return (
<form>

<input type="text" />

<button type="submit">Add</button>

</form>
)
```

Now we will create a state using useState hook. We want to store the input’s data in this state.

```
const [text, setText] = useState("");
```

here, we have text state with initial value of empty string.

Now, everytime user inputs the data, we want this to be stored in our state, so

```
<form>
<input
type="text"
onChange={(e) => setText(e.target.value)}
/>
<button type="submit">Add</button>
</form>
```

So now if we console log the state variable,

```
const [text, setText] = useState("");

console.log(text)
```

we can see how this reflect the change in value of he state.

Step 3

Now we want this input data by user to be part of an array state of Todolist.js

So first we will import TodoForm.js in our Todolist.js

(in Todolist.js)

```
import TodoForm from "./TodoForm";
```

Now we will create a path to collect the data by first passing prop from parent component, Todolist.js to TodoForm.js .

```
<h1>Todo List</h1>
<TodoForm addTodo={addTodo} />
```

We want data arrive from TodoForm.js such a way, we want it to get added in our array, so we will define addToDo such way, that data is added to our empty array.

```
const addTodo = (todo) => {
setTodos([...todos, todo]);
};
```

So this will add our parameter of addTodo to be part of array of todos.

Step 4

Now we will set how data is send from TodoForm.js

So here, we will allow our prop pass through our function.

```
function TodoForm({ addTodo }) {
```

Now we want our input data to be first stored in our state.
So we will create a onSubmit handler in our form.

```
<form onSubmit={handleSubmit}>
```

Now we will define our handleSubmit such a way, it prevent the default refresh of a page on submit.
We will also set the key-value of addTodo we passed in our component.
We want two key for our object, an id because each array item should have unique id.
We will create unique id with javascript Date.now() function. This will return to us millisecond number of time.
We will create a text key which we will value to be state variable text.

```
const handleSubmit = (e) => {
e.preventDefault();
addTodo({
id: Date.now(),
text: text,
})
};
```

Now we will pass a unique key to each list item and also define the value by accessing the text of our addToDo.

(in Todolist.js)

```
h1>Todo List</h1>
<TodoForm addTodo={addTodo} />
<ul>
{todos.map((todo) => (
<li key={todo.id}>
{todo.text}
</li>
```

So now our TodoForm's input data is added to our unordered list as required.

Now problem is our input field after submitting from button holds to its value, we want to reset the input field so user experience the input done when button is clicked.
For this, after we defined the addTodo prop, we can change the state of our variable to be empty.

```
const handleSubmit = (e) => {
e.preventDefault();
addTodo({
id: Date.now(),
text: text,
});
setText("");
};
```

Now we will define our empty string state variable to be value of our input,

```
<input type="text" value={text} onChange={(e) => setText(e.target.value)} />
```

Our TodoList is completed. We can add input field and it get displayed when we submit with button.

But we want with simple single click, we can mark our list item to be completed
and on double click, we want list item to be remove.

This will add more user friendly experience.

Step 5

So on double click, we want to remove the particular list item.
So we will add onDoubleClick event listener to our list element.
We will create a function removeHandler and pass the todo as its parameter, this todo is a parameter of our map() method which is all items in our array.

```
<ul>
{todos.map((todo) => (
<li key={todo.id}
onDoubleClick={() => removeHandler(todo)} >
{todo.text}
</li>
))}
</ul>
```

Now we will define our removeHandler(todo)

So we want our removeHandler function to takes in a object as a parameter. We will change the todos state variable with setTodos such a way, it is creates a new array that do not have same id property as to the parameter that was passed.

```
const removeHandler = (todo) => {
setTodos(todos.filter((item) => item.id !== todo.id));
};
```

The function first uses the "filter" method to create a new array of "todos" that do not have the same "id" property as the "todo" that was passed in. The new array is then set using the "setTodos" function. The effect of this function is that it removes the "todo" object from the array of "todos".

Now we will create a function that will provide a visual cue to the user that the task has been completed by setting the css property of text decoration to be line-through or empty.

So onClick we want to execute a function completehandler that takes paramter todo. This is same todo that was passed as parameter in map() method.

```
<TodoForm addTodo={addTodo} />
<ul>
{todos.map((todo) => (
<li key={todo.id}
onDoubleClick={() => removeHandler(todo)}
onClick={() => completeHandler(todo)}>
{todo.text}
</li>
))}
</ul>
</div>
```

So now we will define a our function completeHandler()

So we want this function to pass an todo object as an argument. The purpose of our function will be to mark task completed or not. So we will create a new array updateTodos. This will fo througt all the exisitng array items of todos using map() method. If the id of a particular item matches the id of the todo passed into the function, the completed property of that item is toggled (from true to false or vice versa) using the `!` operator. The updated item is then returned and added to the new updatedTodos array.
Finally, setTodos is called with the new updatedTodos array to update the state of `todos`.

```
const completeHandler = (todo) => {
let updatedTodos = todos.map((item) => {
if (item.id === todo.id) {
item.completed = !item.completed;
}
return item;
});
setTodos(updatedTodos);
};
```

Now we will sets the style of a text with the React inline style attribute. The style object has a property called text-decoration. If a condition is met and the todo.completed is true, then the value of the text-decoration property will be set to line-through , which indicates that the text will have a strikethrough line. Otherwise, if the todo.completed is false, the text-decoration property will be set to an empty string, meaning that no line will be displayed.

```
<div className="container">
<h1>Todo List</h1>
<TodoForm addTodo={addTodo} />
<ul>
{todos.map((todo) => (
<li key={todo.id}
onDoubleClick={() => removeHandler(todo)}
onClick={() => completeHandler(todo)}
style={{ textDecoration: todo.completed ? "line-through" : "" }} >
{todo.text}
</li>
))}
</ul>
</div>
```